Fatih: Detecting and Isolating Malicious Routers

Alper Tugay Mızrak, Yu-Chung Cheng, Keith Marzullo and Stefan Savage
Dept. of Computer Science and Engineering, University of California, San Diego
{amizrak, ycheng, marzullo, savage}@cs.ucsd.edu

Abstract

Network routers occupy a key role in modern data transport and consequently are attractive targets for attackers. By manipulating, diverting or dropping packets arriving at a compromised router, an attacker can trivially mount denial-of-service, surveillance or man-in-the-middle attacks on end host systems. In this paper, we specify the problem of detecting routers with incorrect packet forwarding behavior and we explore the design space of protocols that implement such a detector. We further present a concrete protocol that is inexpensive enough for practical implementation at scale.

1 Introduction

This paper addresses part of a simple, yet increasingly important network security problem: how to detect the existence of compromised routers in a network and then remove them from the routing fabric. The root of this problem arises from the key role that routers play in modern packet switched data networks. To a first approximation, networks can be modeled as a series of point-to-point links connecting pairs of routers to form a directed graph. Since few endpoints are directly connected, data must be forwarded – hop-by-hop – from router to router, towards its ultimate destination. Therefore, if a router is compromised, it stands to reason than an attacker may drop, delay, reorder, corrupt, modify or divert *any* of the packets passing through. Such a capability can then be used to deny service to legitimate hosts, to implement ongoing network surveillance or to provide an efficient man-in-the-middle functionality for attacking end systems.

Moreover, such attacks are not mere theoretical curiosities, but they are actively employed in practice. Attackers have repeatedly demonstrated their ability to compromise routers, through combinations of social engineering and exploitation of weak passwords or latent software vulnerabilities [2, 21, 27]. One network operator recently documented over 5000 compromised routers as well as an underground market for trading access to them [47]. Once a router is compromised an attacker need not modify the router's code base to exploit its capabilities. Current standard command-line interfaces from vendors such as Cisco and Juniper are sufficiently powerful to drop and delay packets, send copies of packets to a third party, or "divert" packets through a third party and back. In fact, several widely published documents provide a standard cookbook for transparently "tunneling" packets from a compromised router through an arbitrary third-party host and back again – effectively amplifying the attacker's abilities, including arbitrary packet sniffing, injection or modification [18, 45]. Such attacks can be extremely difficult to detect manually, and it can be even harder to isolate which particular router or group of routers has been compromised.

The problem of detecting and removing compromised routers can be thought of as an instance of *anomalous behavior-based intrusion detection*. That is, a compromised router can potentially be identified by correct routers when it deviates from exhibiting expected behavior. This problem can be broken into three distinct subproblems:

1. Traffic validation. Traffic information is the basis of detecting anomalous behavior: given traffic entering a part of the network, and an expected behavior for the routers in the network (i.e. a known routing configuration), anomalous behavior is detected when the monitored traffic leaving one part of the network differs significantly from what is expected. However, implementing such validation practically can be quite tricky and requires tradeoffs between the overhead of monitoring, communication and accuracy.

2. Distributed detection. It is impossible for a single router to establish that its neighbor is anomalous. Any such detection requires synchronizing a collection of traffic information and distributing the results so that anomalous behavior can be detected by *sets* of correct routers.

3. Response. Once a router, or set of routers, is thought to be faulty, the forwarding tables of correct routers must be changed to avoid using those compromised nodes. In addition, over longer time scales an appropriate alert

must be raised so human forensic experts can respond appropriately.

This paper addresses each of these problems in turn. For different threats, we describe a range of appropriate and efficient traffic validation functions. Next we examine how those can be used to build an anomalous behavior detector for compromised routers. Finally, we present a modified link-state routing algorithm that can automatically isolate network paths demonstrating anomalous behavior.

Related Work: There are two threats posed by a compromised router: the attacker may attack by means of the routing protocol (for example, by sending false advertisements) or by having the router violate the forwarding decisions it should make based on its routing tables. The first situation is often referred to as an attack on the *control plane*, while the second is termed an attack on the *data plane*.

The first threat has received, by far, the lion's share of the attention in the research community, perhaps due its potential for catastrophic effects. By issuing false routing advertisements, a compromised router may manipulate how other routers view the network topology, and thereby disrupt service globally. For example, if a router claims that it is directly connected to all possible destinations, it may become a "black hole" for most traffic in the network. While this problem is by no means solved in practice, there has been significant progress towards this end in the research community, beginning with the seminal work of Perlman. In her PhD thesis [36], Perlman described robust flooding algorithms for delivering the key state across any connected network and a means for explicitly signing route advertisements. There have subsequently been a variety of efforts to impart similar guarantees to existing routing protocols with varying levels of cost and protection based on ensuring the authenticity of route updates and detecting inconsistency between route updates [12, 15, 19, 22, 24, 26, 42, 44].

By contrast, the threat posed by subverting the forwarding process has received comparatively little attention. This is surprising since, in many ways this kind of attack presents a wider set of opportunities to the attacker – not only denial-of-service, but also packet sniffing, modification and insertion – and is both trivial to implement (a few lines typed into a command shell) and difficult to detect. This paper focuses entirely on the problem of malicious forwarding.

The earliest work on fault-tolerant forwarding is also due to Perlman [36, 37]. Perlman developed a novel method for robust routing based on source routing, digitally signed *route-setup packets*, reserved buffers. However, many implementation details are left open and the protocol requires higher network level participation to detect anomalies. Several researchers have subsequently proposed lighter-weight protocols for actively probing the forwarding path to test for consistency with advertised routes. Subramanian et al's Listen protocol [44] does this by comparing TCP Data and Acknowledgment packets to provide evidence that a path is part of end-to-end connectivity. This approach only tests for gross connectivity and cannot reveal whether packets have been diverted, modified, created, reordered or selectively dropped. Padmanabhan and Simon's Secure Traceroute [35] achieves a similar goal monitoring the traffic to the intermediate routers. Recently, Avramopoulos *et al.* [3, 4] presents a secure routing a combination of source routing, hop by hop authentication, end-to-end reliability mechanisms and timeouts. But still it has a high overhead to be deployable in modern networks. The approach most similar to our own is the WATCHERS [9, 13] protocol, which detects disruptive routers based on a distributed network monitoring approach and a traffic invariant called conservation of flow. However, the WATCHERS protocol had many limitations in both its traffic validation mechanism and in its control protocol, many of which were documented by Hughes et al. [23]. Many of these weaknesses arose from the absence of a formal specification, a weak threat model and an excessive requirement for per-router state (bounded only by the total size of the network). Herzberg and Kutten [20] presents an abstract model for Byzantine detection of compromised routers based on timeouts and acknowledgments from the destination and possibly from some of the intermediate routers to the source. The requirement of information from intermediate routers offers a trade-off between fault detection time and message communication overhead. This trade-off is analyzed within the given abstract model.

2 System Model

Our work proceeds from an informed, yet abstracted, model of how the network is constructed, the capabilities of the attacker, and the complexities of the traffic validation problem. In this section we describe and motivate the assumptions underlying our model.

Network Model: We consider a network to consist of individual homogeneous routers interconnected via directional point-to-point links. This model is an intentional simplification of real networks (e.g., it does not include broadcast channels or independently failing network interfaces) but is sufficiently general to encompass such details if necessary. Within a network, we presume that packets are forwarded in a hop-by-hop fashion – each router following the directions of a local forwarding table. As well, we assume that these forwarding tables are updated via a distributed link-state routing protocol such as OSPF or IS-IS. This is critical, as we depend on the routing protocol to provide each node with a global view of the current network topology. Finally, we also assume the administrative ability to assign and distribute shared keys to sets of nearby routers. This overall model is consistent with the typical construction of large enterprise IP networks or the internal structure

of single ISP backbone networks, but is not well-suited for networks that are composed of multiple administrative domains using BGP.

At this level of abstraction, we can assume a synchronous network model of synchronized clocks and bounded message delays. Our goal is to extend the routing protocol to detect compromised routers. If the network behaves asynchronously for too long, then the routing tables will be updated, thereby changing the network topology. This assumption is common to all protocols we know of that have addressed the problem of detecting compromised routers.

We define a *path* to be a finite sequence $\langle r_1, r_2, \ldots r_n \rangle$ of adjacent routers. Operationally, a path defines a sequence of routers a packet can follow. We call the first router of the path the *source* and the last router its *sink*; together, these are called *terminal routers*. A path might consist of only one router, in which case the source and sink are the same.

An $x-$*path segment* is a sequence of x consecutive routers that is a subsequence of a path. A *path segment* is an $x-$path segment for some value of $x > 0$. For example, if a network consists of the single path $\langle a, b, c, d \rangle$ then $\langle c, d \rangle$ and $\langle b, c \rangle$ are both 2-path segments, but $\langle a, c \rangle$ is not because a and c are not adjacent.

We do not rely on source routing, as has been done by some work in the past [4, 20, 36]. We do assume some knowledge of the path a packet will take, at least in the stable state. In link state protocols, this can be problematic, because they can take advantage of multiple paths with equal cost for load balancing purposes. Fortunately, the current generation of routers use a deterministic hash algorithm to spread the traffic load across available interfaces (eg, Cisco Express Forwarding [14] and Juniper routers with an Internet Processor ASIC [25]). Thus, a router can predict the path a packet will take in the stable state based on its own routing tables and the hash functions.

Threat Model: We assume that attackers can compromise one or more routers in a network and may even compromise sets of adjacent routers as well. In general, we parameterize the strength of the adversary in terms of the maximum number of adjacent routers along a given path that can be compromised. However, we assume that between any two uncompromised routers that there is sufficient path diversity that the malicious routers do not partition the network. In some sense, this assumption is pedantic since it is impossible to guarantee any network communication across such a partition. Another way to view this constraint is that path diversity between two points in the network is a necessary, but insufficient, condition for tolerating compromised routers. What we are proposing is a set of protocols that offer the sufficiency condition in the presence of the necessary diversity.

Recently, Teixeira et al. [46] empirically measured path diversity in ISP networks and found that multiple paths between pairs of nodes were common. Similarly, many enterprise networks are designed with such diversity in order to mask the impact of link failures. Consequently, we believe that this assumption is reasonable in practice. It is worth noting however, that this diversity usually does not extend to individual local-area networks – single workstations rarely have multiple paths to their network infrastructure. Consequently, the fate of individual hosts and of the router to which they are connected, are directly intertwined in practice. So, we assume that a terminal router is not faulty with respect to traffic originating or being consumed by that router.

Since link-state protocols operate by periodically measuring and disseminating information, we assume a synchronous system. The failure of a router is defined in terms of an interval of time, which in practice corresponds with a period of time during which traffic measurements are made. Specifically, a router r is *traffic* faulty with respect to a path segment π during τ if π contains r and, during the period of time τ, r exhibits anomalous behavior with respect to forwarding data that traverses π. For example, router r can selectively alter, misroute, drop, reorder, or delay the data that flows through π, and it can fabricate new data to send along π such that the packets, if they were valid, would have been routed through π.

3 Traffic Validation

The first problem we address is *traffic validation*: what information is kept about packet traffic and how it is used to determine that a router has been compromised.

We assume that a compromised router can arbitrarily alter the forwarding behavior of that router. For example, a compromised router can drop or modify selected (or all) packets, or divert them to other routers. However, given the distributed nature of packet forwarding, such bad behavior can be detected. For example, suppose traffic traverses a path $\langle r_1, r_2, r_3 \rangle$ and router r_2 modifies this traffic. If routers r_1 and r_3 are not compromised, then one could simply compare what r_1 sent to r_2 and what r_3 received from r_2 to detect r_2's behavior.

At an abstract level, we represent traffic validation mechanisms as a predicate *TV*(π, *info*(r_i, π, τ), *info*(r_j, π, τ)) where:

- π is a path segment $\langle r_1, r_2, \ldots r_x \rangle$ whose traffic is to be validated between routers r_i and $r_j \in \pi$;
- *info*(r, π, τ) is information about traffic that router r forwarded to along π over some time interval τ.
- If routers r_i and r_j are not faulty, then *TV*(π, *info*(r_i, π, τ), *info*(r_j, π, τ)) evaluates to *false* iff π contains a router that was faulty in forwarding traffic along π during τ.

Implementing a traffic validation mechanism is a tricky engineering problem. The simplest, and most precise, representation of *info*(r, π, τ) is a complete copy of the packets sent, with each packet tagged with the time it was forwarded. However, the storage requirements to buffer these packets and the bandwidth consumed by resending them, make this approach impractical at best. In practice, implementing traffic validation is a tradeoff between accuracy and overhead. For example, sampling can be used to decrease overhead, but depending on how sampling is done and the duration of the attack, accuracy may be reduced. The overhead-accuracy tradeoff also depends on what limits one might place on the kind of bad behavior a compromised router can exhibit.

Similarly, TV could be implemented simply using equality; *info*$(r_i, \pi, \tau) =$ *info*(r_j, π, τ). However, real networks occasionally lose packets due to congestion, reorder packets due to internal multiplexing, and corrupt packets due to interface errors. Consequently, TV needs to be somewhat more sophisticated to accommodate this abnormal, but non-malicious behavior. Thus, implementing traffic validation involves striking a balance between the acceptable number of false positives and false negatives.

Note that we have already made one engineering decision: we analyze aggregate traffic rather than individual packets. This is in contrast to some prior work, e.g. Perlman [36], Herzberg *et al.* [20], and Avramopoulos *et al.* [4]. Doing so amortizes the monitoring overhead over many packets; if aggregation is not done then the computation and storage overhead is prohibitively large. It also makes it feasible to apply a threshold mechanism to distinguish between acceptable bad behavior (e.g. small amounts of packet loss and reordering) and malicious behavior.

3.1 Characterizing Traffic

While the most precise description of traffic sent into the network is an exact copy of that traffic, many characteristics of the traffic can be summarized far more concisely. In particular, if we concentrate on particular threats – ways in which a malicious entity might alter the traffic – we can limit our effort to detecting just those actions.

One limitation of previous work has been that they assumed limited kinds of threats. For example, `WATCHERS` [23] and Herzberg *et al.* [20] effectively detect only packet losses that result from a compromised router. In all of the previous work, reordering attacks have been ignored, even though reordering of packets can effect TCP performance tremendously [5, 6]. It would not be hard to extend Secure Traceroute [35] to include this kind of attack, but it can not be done with the protocols of Perlman [36], Herzberg *et al.* [20] and Avramopoulos *et al.* [4] since they monitor a single packet at each round. We instead divide up arbitrary behavior into five different threats:

- *Packet loss*. A compromised router can drop any subset of the packets. As per Almes et al. [1] loss can be measured as the mount of data arriving at the sink of path segment subtracted from the amount of data sent from its source.

- *Packet fabrication*. A compromised router can generate packets and inject them into the traffic stream. This can be measured as the number packets which are reported at the sink of a packet segment but not monitored as being sent by its source.

  Misrouting packets can be considered an instance of both packet loss and packet fabrication.

- *Packet modification*. One can consider this threat as a combination of packet loss and fabrication, but it may not be detectable by simply comparing the number of packets arriving at the sink with the number sent from the source. Instead, some summary of the content needs to be maintained, and one measures the number of modified packets.

- *Packet reordering*. A compromised router can reorder packets. Doing this can lead to performance problems or, in the extreme, denial of service. There are many reasonable and incompatible methods of measuring the amount of reordering, e.g. [5, 6, 31, 38]. We use the definition from [38] because of its simplicity: given a transmitted stream S and a received stream F, remove from both all lost, fabricated and modified packets. Then, find the longest common subsequence ℓ between these modified S and F. The amount of reordering is defined as $|S| - |\ell|$.

- *Time behavior*. A compromised router can delay traffic. Like reordering, doing this can lead to performance problems or, in the extreme, denial of service. There are simple metrics one can use, such as the first n moments of the inter-packet delay distribution. However, such metrics are notoriously sensitive in packet networks; we believe this area requires more research.

These threats completely cover the set of bad behaviors a router can exhibit in forwarding data. When all of these metrics are zero, then no router is forwarding traffic in a faulty manner.

3.2 Traffic Summary Functions

We have implemented three traffic summary functions, each reflecting a different tradeoff among accuracy, completeness, and threat. They are:

Conservation of flow: Our first summary function resembles the "conservation of flow" approach used by the `WATCHERS` protocol [9]. The traffic information

info(r_i, π, τ) collected by router r_i consists of two counters, each counting the number of packets in one of the two directions of traffic along the path π. Periodically, r_i passes these counters to other routers collecting information about π. Traffic validation is done by comparing the values of these counters. In general, this is a fragile summary function because it only detects actions that cause packet losses, and it assumes that malicious routers cannot fabricate packets to "fudge" the counts appropriately. However, it is extremely cheap to implement, both in the per packet cost and in the associated overhead to communicate the traffic information among routers.

Conservation of content: To detect modification of packets, we can use a fingerprint (that is, a hash), of the payload in place of a simple counter. Analogous to conservation of flow, each router then periodically communicates a set of packet fingerprints with other routers for traffic validation, which is calculated via set difference. In addition to detecting packet modification, this approach also detects packet loss, packet fabrication, and misrouting.

One technical difficulty with this approach is that packets are naturally modified as they traverse routers. In particular, the TTL field in the IP header is decremented and the IP header checksum is updated. We address this by having each router compute fingerprints based on the TTL value and checksum at one end of the path π.

One downside to this approach is that it requires storing and communicating a fingerprint for each packet forwarded by a router. This is a significant overhead. One can save significant space and bandwidth by using more sophisticated algorithms for calculating set differences. The simplest approach is to simply use Bloom filters [8] to represent the set of fingerprints. One can then use the population of the bitwise difference between the filters to calculate the size of the set difference. This approach is far cheaper to implement, but comes at some expense in accuracy. More problematic is that it is difficult to know in advance the appropriate parameters for the Bloom filter. A too-small filter can result in significant errors in estimation. A more promising approach is to leverage distributed set reconciliation algorithms [29]. This approach has greater computation overhead than Bloom filters, but can be *optimal* in bandwidth utilization.

Conservation of content order: Packet reordering can be detected by maintaining ordered lists of packet fingerprints rather than simply sets. Like with conservation of content, this can result in a significant storage overhead. Traffic validation can then be done using the metric defined in Section 3.1. This has a higher overhead than simply computing the size of the set difference. We have not yet looked into methods that would reduce the storage or computational overheads.

4 Distributed Detection

The second problem is synchronizing the collection of traffic information and distributing the results for detection purposes. The solution to this problem is a protocol, and so we consider specifications of the problem as well as implementation.

Since routers collect the information upon which traffic validation is based, there will be some uncertainty in determining which router is faulty. For example, consider traffic that traverses a path containing the sequence of routers $\langle r_1, r_2, r_3 \rangle$. Suppose router r_3 receives 10 packets from r_2 and r_1's traffic information state that r_1 sent 20 packets to r_2. There's no way for r_3 to determine whether r_2 did indeed drop 10 packets, or whether r_1 is misreporting the traffic.

We cast the problem as a failure detector with *accuracy* and *completeness* properties. This failure detector reports suspicions as path segments, which are sequences of adjacent routers. More specifically, a failure detector reports a path segment π if it suspects a router in π is forwarding traffic along π in a faulty manner. A failure detector also has a *precision*, which is the maximum length of a path segment it suspects.

4.1 Specification

Due to the nature of the problem, our specification is more complex than the typical accuracy and completeness properties of an imperfect failure detector [10]). Since this detector is based on evaluating traffic collected over a period of time, we have the failure detector report pairs (r, τ), which means that r was suspected as being faulty during the time interval τ. A perfect failure detector would implement the following two properties:

- *Accuracy (tentative):* A failure detector is *accurate* if, whenever a correct router suspects (r, τ), then r was faulty during τ.
- *Completeness (tentative):* A failure detector is *complete* if, whenever a router r is faulty at some time t, then all correct routers eventually suspect (r, τ) for some τ containing t.

As noted above, though, our failure detector is based on traffic information collected by untrustworthy routers. This results in detections that are imprecise: we may not be able to pin down which router is compromised. So, we have the failure detector return a pair (π, τ) where π is a path segment. This also allows us to give more information: we restrict the detection to a router being faulty with respect to forwarding traffic along π.

- *a–Accuracy:* A failure detector is *a–Accurate* if, whenever a correct router suspects (π, τ), then $|\pi| \leq a$ and some router $r \in \pi$ was faulty in π during τ.

- *a–Completeness (tentative):* A failure detector is *a–Complete* if, whenever a router r is faulty at some time t, then all correct routers eventually suspect (π, τ) for some path segment $\pi : |\pi| \leq a$ such that r was faulty in π at t, and for some interval τ containing t.

Note that a faulty router can report an incorrect value for the traffic that traversed a path as well as alter this traffic. We use the term *t-faulty* (that is, *traffic* faulty) to indicate a router that alters traffic and the term *p-faulty* (that is, *protocol* faulty) to indicate a router that misreports traffic. A *faulty* router is one that is t-faulty, p-faulty or both. As before, we will add the phrase "in π" to indicate that the faulty behavior is with respect to traffic that transits the path π. Thus, the a–Accuracy requirement can result in a detection if a router is either p-faulty or t-faulty.

Distinguishing between p-faulty and t-faulty behavior is useful because, while it is important to detect routers that are t-faulty, it isn't as critical to detect routers that are only p-faulty: routers that are only p-faulty are not altering the traffic flow. Hence, we weaken a–Completeness:

- *a–Completeness:* A failure detector is *a–Complete* if, whenever a router r is t-faulty at some time t, then all correct routers eventually suspect (π, τ) for some path segment $\pi : |\pi| \leq a$ such that r was t-faulty in π at t, and for some interval τ containing t.

One can't depend on faulty servers to detect faulty servers. Hence, failure detection is influenced more by the maximum number of adjacent faulty routers rather than the total number of faulty routers. So, we impose an upper bound *bad*(k) on the number of adjacent faulty routers. For example, if *bad*(3) holds, then there can be no more than 3 adjacent faulty routers in any path.

Making a *bad*(k) assumption has an effect on failure detection. Allowing compromised routers to be adjacent complicate detection further, since they can cooperate to hide the evidence that a router is faulty. For example, assume that *bad*(3) holds, and consider a path $\langle r_1, r_2, r_3, r_4, r_5, r_6, r_7 \rangle$ in which r_3, r_4 and r_5 are faulty. Suppose that over an interval τ, r_1 through r_5 report having forwarded 100 packets that were to traverse this path, and r_6 and r_7 reports only 20 such packets. Let r_1 obtain these counters. It could be the case that r_4 dropped the traffic, and r_3 and r_5 misreported the traffic in an effort to hide the fact that r_4 is faulty. From r_1's point of view, however, something is wrong with either r_5 or r_6, since the counters indicate that traffic has disappeared between them. To satisfy a–Completeness, p_1 needs to detect any possible routers that could have led to this traffic discrepancy. So, the failure detector at r_1 could report that the path segment $\langle r_3, r_4, r_5, r_6 \rangle$ contains a faulty router, since if r_5 is faulty then r_3 and r_4 could be as well, given *bad*(3). That is, we could have the failure detector report a $k+1$-length path segment to accommodate the fact that *bad*(k) holds.

Another way to accommodate this kind of scenario is to weaken a–Completeness. In the example just given, we could have r_1 just suspect the path segment $\langle r_5, r_6 \rangle$. A countermeasure protocol would stop routing data through this path, and if r_3 or r_4 continue to behave in a faulty manner, then they would be suspected later. We formalize this approach by defining a condition we call being *fault connected*. Given a path segment π and an interval τ, we say that a router r is fault connected to router s with respect to π if both r and s are in π, and all of the routers between s and r are faulty in π during τ. Trivially, any router r is fault connected to itself even if r is correct. We then weaken a–Completeness again:

- *a–FC Completeness:* A failure detector is *a–FC Complete* if, whenever a router r is t-faulty at some time t, then all correct routers eventually suspect (π, τ) for some $\pi : |\pi| \leq a$ and some τ containing t such that there is a router r' that was faulty in π at time t' in τ and is fault-connected to r.

The choice between the two completeness properties is not obvious. One would expect the precision obtainable under a–FC Completeness to be better, but it could increase the latency in detecting some t-faulty routers. If the value of k for which *bad*(k) holds is not large, then a–Completeness is probably a better choice because of its simplicity.

Since we are assuming arbitrarily faulty routers, we have to allow faulty router to suspect correct routers. We address this issue in the countermeasure protocol: as with WATCHERS, the only suspicion that elicits a countermeasure is when a router r suspects a path segment that is adjacent to r. In this case, the countermeasure protocol will cause r not to route traffic along the suspected path segment. A faulty router can already drop packets, and so allowing a faulty router to break links with its neighbor (as a result of faulty suspicions) adds no further disadvantage.

Using this terminology, Perlman [36] protocol is M–Accurate and M–Complete, where M is the maximum length of a path between any pair of routers. All other prior protocols we know of are 2–Accurate. However, the detection is attained at only the source router in Perlman [36], Avramopoulos *et al.* [4] and Secure Traceroute [35]. If other routers learn of the fault from the detecting router r, then such a router has to consider the possibility that r is p-faulty. `WATCHERS` [23] is neither Complete nor FC Complete, due to a flaw, which we have shown in [30]. The claims in [35] indicate that Secure Traceroute is 2–Accurate and 2–FC Complete. But, the description of the protocol (especially on how the overhead is reduced by pretending to monitor all prefixes of a path segment from a source to a destination) is not detailed enough for us to be confident of the

actual accuracy and completeness of the protocol. Herzberg *et al.* [20] and Avramopoulos *et al.* [4] protocols are 2–FC Complete[1].

4.2 Π_{k+2}: A $(k+2)$–Complete and $(k+2)$–Accurate Protocol

Given a complete and accurate traffic validation mechanism, it isn't hard to come up with a detection protocol. A simple approach is to use consensus to distribute the traffic information and then have each router use TV to implement its failure detector. In particular, each router r monitors a set P_r of path segments. This set consists of all $(k+2)-$path segments containing r and all $x-$path segments, $3 \leq x < k+2$ whose ends are terminal routers. For each path segment $\pi \in P_r$, r synchronizes with the other routers in π and collects information for the same traffic passing through π for an agreed-upon interval τ. Periodically, r sends that traffic information (digitally signed to prevent modification) to all routers in π using consensus. Each router then applies TV to the traffic data. This results in a protocol that we call Π_2, and do not develop it further here. In [30], we have shown that Π_2 is 2–Accurate and 2–FC Complete. It has considerable requirements in terms of the amount of traffic information each router needs to collect and in synchronization. We have instead implemented a simpler protocol that has fewer requirements but is less precise.

```
failure detector()
  cobegin
    for each path segment π ∈ P_r:
      suspect_r^τ[] = { } // the set of unreliable path segments
                          // that r detects during τ
      while (true) {
        synchronize with the router r' at other end of π;
        collect traffic information info(r, π, τ) about π
                for an agreed-upon interval τ;
        exchange [info(r, π, τ)]_r and [info(r', π, τ)]_r' with r'
                through π within μ timeout interval;
        if (exchange is failed or
                ¬TV(π, info(r, π, τ), info(r', π, τ))) then {
          suspect_r^τ[] = suspect_r^τ[] ∪ {⟨π⟩};
          reliable broadcast ([π]_r);
        }
      }
  coend
```

Figure 1. Π_{k+2}

The idea is to apply TV just for the end nodes of each path segment in P_r. For example, consider the 4-path segment $\pi = \langle r_1, r_2, r_3, r_4 \rangle$ where r_1 and r_4 are not faulty. Let *info*(r_1, π, τ), *info*(r_4, π, τ) be the traffic information that router r_1 and r_4 collect during τ. If at least one of the other routers is t-faulty with respect to π during this interval, then *TV*(π, *info*(r_1, π, τ), *info*(r_4, π, τ)) will be false. In this case, r_1 suspects $\langle r_2, r_3, r_4 \rangle$. Similarly r_4 suspects $\langle r_1, r_2, r_3 \rangle$.

For this protocol, a router need not monitor as many path segments as with Π_2. Instead, a router needs only keep track of each x-path segment of which it is one of the end nodes, for some value of x. The number of x-path segments can grow very quickly with increasing x, and so x should be as small as possible. It must be large enough so that any sequence of faulty routers will be surrounded by correct routers, as this is necessary to detect faulty behavior.

If we assume that *bad*(k) holds, then the minimum value of x satisfying the above constraint is $k+2$. However, monitoring only $k+2$-path segments is not sufficient. A trivial reason this is so is that not all path segments need be $k+2$ long. A more substantial reason is that compromised routers may hide another router's bad behavior. For example, given that *bad*(2) holds, consider the 4-path segment $\pi = \langle r_1, r_2, r_3, r_4 \rangle$ where r_1 and r_3 are correct and r_2 and r_4 are faulty. In this case, r_1 and r_4 monitors π but r_4 can hide the fact that r_2 is t-faulty by simply sending traffic information to r_1 such that *TV*(π, *info*(r_1, π, τ), *info*(r_4, π, τ)) holds. If r_1 were to instead also monitor the path $\langle r_1, r_2, r_3 \rangle$, then r_1 could detect r_2's faulty behavior. So, it is necessary for a router r to monitor all x-path segments for $3 \leq x \leq k+2$ of which r is an end.

For each path segment $\pi \in P_r$, r synchronizes with the other end router of π and collect information for the traffic passing through π during an agreed-upon interval τ. Router r then exchanges digitally signed traffic information with the router r' on the other end. If the exchange operation fails within a pre-specified timeout interval μ, or if r finds *TV*(π, *info*(r, π, τ), *info*(r', π, τ)) is false, then there is at least one faulty router in π during τ. In particular, either r' is p-faulty or some router in π is t-faulty. So, r detects $\pi - \langle r \rangle$. However, when it announces this detection to the other routers, a correct router receiving this information suspects π since r might be faulty. For simplicity, we also have router r suspect π.

Π_{k+2} is given in Figure 1. In [30], we have shown that Π_{k+2} is $(k+2)$–Accurate and $(k+2)$–Complete.

5 Response

Once a path segment π is detected as containing compromised routers, some countermeasure should be taken. Of course, the most important countermeasure is to log the suspicion and alert the administrator of the affected routers. Less obvious is how the routers should react to a detection.

Suppose that some path segment π is detected. An obvious countermeasure would be to remove all of the routers in π form the routing fabric. By doing this, we are avoiding using any router that has been suspected of being com-

[1]However, the detection has been done only at the source router. Not every router in the network agree on the detection as we specified above.

promised. Doing this, though, could also have a serious, and perhaps unnecessarily high, impact on network performance. A less aggressive countermeasure would be to only remove the path segment π from the routing fabric. In doing so, we may allow compromised routers to keep forwarding packets, but only along paths over which no faulty behavior has been observed.

We have chosen the second approach because of its less disruptive behavior. If a router is disrupting traffic along several paths, then all of these paths will be suspected and removed. In the worst case, a router may iteratively attack each path through it, which will eventually isolate the router from the fabric.

6 Analysis of the Protocols

In this section, we consider the overhead of our approach.

Fingerprinting: Like any protocol that detects packet alteration [4,35,36], our *Conservation of content* and *Conservation of content order* traffic summary functions compute a fingerprint for each packet. To be practical, computing a fingerprint must have a low overhead. One possibility for fingerprinting is CRC32. This is already computed by most network interfaces at line rate and is a good uniform hash function. However, it is reversible and an adversary could create modified packets that have the same hash. In some circumstances, such as a compute limited adversary, it might serve as a good fingerprinting function.

We implemented fingerprinting using UHASH which is an unkeyed version of the UMAC algorithm [7]. This algorithm allows a trade-off between security and performance and is designed to support parallel implementations. [7,39] demonstrated UHASH performance of over 1Gbps on a 700Mhz Pentium III processor computing a 4 byte hash value and delivering a 2^{-30} forging probability (which is more than sufficient for our application). In hardware, of course this performance could be increased still further.

Between software and hardware are emerging network processors which are designed to perform highly parallel actions on data packets [40]. Recently, Feghali *et al.* [17] described an implementation of DES [33] and AES [34],[2] on the Intel IXP28xx network processors that was able to keep pace with a 10Gigabit/sec forwarding rate. Since DES and AES encryption algorithms are significantly more expensive to compute than functions such as UHASH, we can infer that it is possible to compute these packet fingerprints up to 10Gbps.

However, if there are insufficient computational resources, one can easily tradeoff accuracy for overhead by subsampling the packets to be considered. As in Duffield and Grossglauser's Trajectory Sampling [16], if the same random hash function is used to subsample packets at each end of a path segment, then each router should observe the same subset of packets. Further each pair of routers is free to select such sampling functions independently and need not rely on a global secret.

State Size: One of the most important factors in terms of a protocol being practical is the size of the state at each router that needs to be maintained. Assume that we wish to provide fault-tolerant forwarding between each source and destination pair in the network. Let N be the number of routers in the network, R be the maximum number of links incident on a router and k is the value used in the *bad*(k) assumption. Perlman [36], Herzberg *et al.* [20] and Avramopoulos *et al.* [4] require $O(N^2)$ state, since each individual router maintains state for each (source, destination) pair. `WATCHERS` [23] reduces the state requirement to $O(RN)$, where each individual router keeps seven counters for each its neighbors for each destination. In Secure Traceroute [35], if a router monitors paths to all destinations, then $O(N)$ space is required (assuming that there is only one path in active use between a given source and destination).

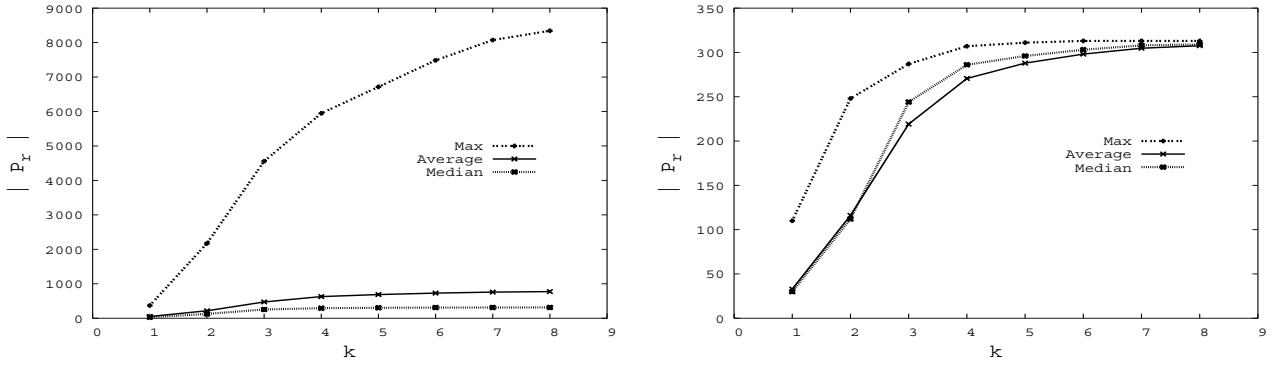
Our protocols require a router r to record state for each path segment P_r that it monitors. By construction, this is $O(k \times R^{k+1})$ for Π_2 and $O(\min\{R^{k+1}, N\})$ for Π_{k+2}. In practice, though, we expect $|P_r|$ to be much smaller. For example, we have examined two network topologies, Sprintlink and EBONE, that were measured by the Rocketfuel project [43]. We counted the number of distinct path segments that a router monitors for different values of k in *bad*(k) assumption. For instance, the Sprintlink network consists of 315 routers and 972 links. On the average, a router has 6.17 links, and the maximum number of links that a router has is 45. In Figure 2, the maximum, average and median of $|P_r|$ that a router monitors is given for this network for Π_2 and Π_{k+2}. For this network (as well as for EBONE), the empirical results are much smaller than the theoretical upper bounds.

As a point of comparison, for the Sprintlink network, on average a router under `WATCHERS` would maintain approximately 13,600 counters, and the maximum number of counters that a router maintains is 99,205. With our conservation of flow traffic summary function[3] for Π_{k+2}, a router maintains 232 counters on average and 496 in the worst case if *bad*(2) holds. Even with the weak assumption of *bad*(7), each router maintains 616 counters on average and 626 in the worst case.

Synchronization: Protocols that validate aggregate traffic requires synchronization to agree on a time interval to collect traffic information. `WATCHERS` synchronizes all

[2] DES and AES are two well known encryption algorithms providing confidentiality.

[3] 2 counters per path segment, one for each direction.



(a) Π_2 (b) Π_{k+2}

Figure 2. Based on *bad*(k); maximum, average and median size of P_r that a router monitors in Π_2, and Π_{k+2}.

routers using a snapshot algorithm [11]. In Π_2, for each path segment π in P_r, a router r synchronizes with all the routers in π to agree on when and for how long the next measurement interval τ will be. It would probably be more efficient, though, to have all the routers in the network synchronize with each other instead of having many more, smaller synchronization rounds. Perfect synchronization would not be necessary in practice, since the traffic validation function TV could be written to accommodate a small skew. The synchronization requirements for Π_{k+2} are lower than for Π_2. As for each path segment π that a router r monitors, r needs to agree with only the other end router r' of π. This is the same requirement for Secure Traceroute.

Information dissemination: Π_2 requires each router in path segment π to reach consensus about the traffic information collected over π during a time interval τ. To do so necessitates digitally signing the traffic information, since otherwise the replication is not high enough for consensus to be solvable. Thus, there is an issue of *key distribution* depending on the cryptographic tools that are used. Finally, there must be enough path connectivity among the routers to support consensus [28]. For Π_{k+2} in order to exchange traffic information, neither Consensus nor the *good neighbor* condition of `WATCHERS` is required (*good neighbor* requires that each compromised router is adjacent to a non-compromised router). Furthermore, with Π_{k+2} the end routers can use the same path segment they are monitoring to exchange traffic information. This is because if an intermediate router were to fail to forward the information, then one end would detect it, which would lead to the path segment being suspected. To avoid impersonation attacks, Π_{k+2} requires authentication. Meanwhile, the final reliable broadcast of both Π_2 and Π_{k+2} can be done as part of the LSA distribution of the link state protocol.

Multicast: Most of the protocols mentioned above can be extended to support multicast, assuming that all routers agree on the network topology and the multicast tree, e.g MOSPF [32]. `WATCHERS` is an exception, since multicast is a clear violation of conservation of flow principle.

Fragmentation: Since fragmentation generally conserves the size of flows, `WATCHERS` handles it naturally, but protocols validating traffic using conservation of content or order are vulnerable. However, we have chosen to ignore this issue since packet fragmentation is rare in practice [41].

7 Fatih: Prototype System

We have implemented a prototype system, called Fatih, that incorporates our approach into a Linux 2.4-based router platform running OSPF. We have implemented many of the traffic validation mechanisms mentioned in Section 3, including conservation of flow, content and content order, as well as the Π_{k+2} distributed detection algorithm explained in Section 4.2. Fatih is able to detect and isolate a range of malicious router actions with acceptable overhead and complexity, and with very small changes to the underlying link state protocols. While our system is only a prototype and there are still issues to address, we believe that our work shows that this approach has practical promise.

8 Conclusion

In this paper, we motivated and specified the problem of detecting routers with incorrect packet forwarding behavior and provided a general framework for formally analyzing protocols addressing this problem. Our approach is to separate the problem into three subproblems: (1) efficiently characterizing traffic to detect deviations, (2) synchronizing routers to collect and distribute these traffic characterizations, (3) taking countermeasures when significant traffic deviations are detected. We presented a concrete protocol, Π_{k+2}, that is inexpensive enough for practical implementation at scale. We believe our work is an important step in being able to tolerate attacks on key network infrastructure components.

References

[1] G. Almes, S. Kalidindi, and M. Zekauskas. A one-way packet loss metric for IPPM. *RFC 2680*, Sept. 1999.

[2] X. Ao. Report on DIMACS Workshop on Large-Scale Internet Attacks, Sept. 2003.

[3] I. Avramopoulos, H. Kobayashi, R. Wang, and A. Krishnamurthy. Amendment to: Highly secure and efficient routing, Feb. 2004. Amendment.

[4] I. Avramopoulos, H. Kobayashi, R. Wang, and A. Krishnamurthy. Highly secure and efficient routing. In *Proceedings of INFOCOM 2004 Conference*, March 2004.

[5] J. Bellardo and S. Savage. Measuring packet reordering. In *ACM SIGCOMM Internet Measurement Workshop(IMW02)*.

[6] J. C. R. Bennett, C. Partridge, and N. Shectman. Packet reordering is not pathological network behavior. *IEEE/ACM Transactions on Networking (TON)*, 7(6):789–798, 1999.

[7] J. Black, S. Halevi, H. Krawczyk, T. Krovetz, and P. Rogaway. UMAC: Fast and secure message authentication. *Lec. Notes in CS*, 1666:216–233, 1999.

[8] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Comm. of the ACM*, 13(7):422–426, July '70.

[9] K. A. Bradley, S. Cheung, N. Puketza, B. Mukherjee, and R. A. Olsson. Detecting disruptive routers: A distributed network monitoring approach. In *Proc. of the IEEE Symposium on Security and Privacy*, pages 115–124, May 1998.

[10] T. D. Chandra and S. Toueg. Unreliable failure detectors for reliable distributed systems. *Journal of the ACM*, 43(2):225–267, 1996.

[11] K. M. Chandy and L. Lamport. Distributed snapshots: determining global states of distributed systems. *ACM Transactions on Computer Systems*, 3(1):63–75, 1985.

[12] S. Cheung. An efficient message authentication scheme for link state routing. In *ACSAC*, pages 90–98, 1997.

[13] S. Cheung and K. Levitt. Protecting routing infrastructures from denial of service using cooperative intrusion detection. In *New Security Paradigms Workshop*, 1997.

[14] Cisco Systems. Load balancing with cisco express forwarding. http://www.cisco.com/warp/public/cc/pd/ifaa/pa/much/prodlit/loadb_an.pdf.

[15] Y. Cosendai, M. Dacier, and P. Scotton. Intrusion detection mechanism to detect reachability attacks in PNNI networks. In *Recent Advances in Intrusion Detection*, 1999.

[16] N. G. Duffield and M. Grossglauser. Trajectory sampling for direct traffic observation. In *SIGCOMM'00*, pages 271–282.

[17] W. Feghali, B. Burres, G. Wolrich, and D. Carrigan. Security: Adding protection to the network via the network processor. *Intel Technology Journal*, 06:40–49, Aug. 2002.

[18] Gauis. Things to do in Ciscoland when you're dead, Jan. '00.

[19] M. T. Goodrich. Efficient and secure network routing algorithms, Jan 2001. Provisional patent filing.

[20] A. Herzberg and S. Kutten. Early detection of message forwarding faults. *SIAM J. Comput.*, 30(4):1169–1196, 2000.

[21] K. J. Houle, G. M. Weaver, N. Long, and R. Thomas. Trends in denial of service attack technology. Technical report, CERT Coordination Center, Oct. 2001.

[22] Y.-C. Hu, A. Perrig, and D. B. Johnson. Ariadne: A secure on-demand routing protocol for ad hoc networks. In *The 8th ACM Int. Conf. on MobiCom*, Sep 2002.

[23] J. R. Hughes, T. Aura, and M. Bishop. Using conservation of flow as a security mechanism in network protocols. In *IEEE Symp. on Security and Privacy*, pages 132–131, 2000.

[24] Y. Jou, F. Gong, C. Sargor, X. Wu, S. Wu, H. Chang, and F. Wang. Design and implementation of a scalable intrusion detection system for the protection of network infrastructure. In *DISCEX'00 Proceedings*, volume 2, pages 69–83, 2000.

[25] Juniper Networks. JUNOS 6.4 Routing Protocols Configuration Guide. http://www.juniper.net/techpubs/software/junos/junos64/swconfig64-routing/html/.

[26] S. Kent, C. Lynn, J. Mikkelson, and K. Seo. Secure Border Gateway Protocol (Secure-BGP). *IEEE Journal on Selected Areas in Communications*, 18(4):582–592, Apr. 2000.

[27] C. Labovitz, A. Ahuja, and M. Bailey. Shining light on dark address space. Technical report, Arbor Networks, Nov. 2001.

[28] L. Lamport, R. Shostak, and M. Pease. The Byzantine generals problem. *ACM Trans. on Programming Languages and Systems*, 4(3):382–401, 1982.

[29] Y. Minsky, A. Trachtenberg, and R. Zippel. Set reconciliation with nearly optimal communication complexity. In *Int. Symp. on Information Theory*, page 232, June 2001.

[30] A. T. Mizrak, K. Marzullo, and S. Savage. Detecting malicious routers. Technical Report CS2004-0789, UCSD, 2004.

[31] A. Morton, L. Ciavattone, G. Ramachandran, S. Shalunov, and J. Perser. Packet reordering metric for IPPM, Mar. 2003.

[32] J. T. Moy. Multicast extensions to OSPF. *RFC 1584, IETF*, Mar. 1994.

[33] National Institute of Standards and Technology. Data encryption standard. *FIPS PUBS 46-3*, Oct. 1999.

[34] National Institute of Standards and Technology. Advanced encryption standard. *FIPS PUBS 197*, Nov. 2001.

[35] V. N. Padmanabhan and D. Simon. Secure traceroute to detect faulty or malicious routing. *SIGCOMM Comp. Comm. Review*, 33(1):77–82, 2003.

[36] R. Perlman. *Network Layer Protocols with Byzantine Robustness*. PhD thesis, MIT LCS TR-429, Oct. 1988.

[37] R. Perlman. *Interconnections: Bridges and Routers*. Addison Wesley Longman Publishing Co. Inc., 1992.

[38] D. Pullin, A. Corlett, B. Mandeville, and S. Critchley. Packet reordering: The minimal longest ascending subsequence metric, Feb. 2002.

[39] P. Rogaway. UMAC Performance (more). www.cs.ucdavis.edu/ rogaway/umac/2000/perf00bis.html.

[40] N. Shah. Understanding network processors. Master's thesis, University of California, Berkeley, September 2001.

[41] C. Shannon, D. Moore, and K. C. Claffy. Beyond folklore: observations on fragmented traffic. *IEEE/ACM Trans. Netw.*, 10(6):709–720, 2002.

[42] B. R. Smith and J. Garcia-Luna-Aceves. Securing the border gateway routing protocol. In *Proc. Global Internet'96*.

[43] N. Spring, R. Mahajan, and D. Wetherall. Measuring ISP topologies with Rocketfuel. In *Proc. of ACM/SIGCOMM*, pages 133–145, 2002.

[44] L. Subramanian, V. Roth, I. Stoica, S. Shenker, and R. Katz. Listen and Whisper: Security Mechanisms for BGP. In *Proc. of NSDI*, Mar. 2004.

[45] D. Taylor. Using a compromised router to capture network traffic, July 2002. Unpublished Technical Report.

[46] R. Teixeira, K. Marzullo, S. Savage, and G. M. Voelker. In search of path diversity in ISP networks. In *Proc. of the ACM/SIGCOMM IMC*, pages 313–318, 2003.

[47] R. Thomas. ISP Security BOF, NANOG 28, June 2003.